SOAR USER'S MANUAL

Marshall A. Soares

May 7, 1997

TR UUCS-97-006

Abstract: ***The development of simulation and test stimulus and checking of circuits with that stimulus is the source or many circuit bugs. The SOAR conversion package is a C library that generates the stimuli for gate-level simulation, circuit simulation and integrated circuit test. The conversion package links to the designer's circuit emulation code through data structures. During the emulator execution, the conversion library outputs the stimuli as a byproduct of the emulation. Files contain the vector, voltage, and timing definitions for the conversion library. This work describes the conversion library and methods to create stimulus for Viewlogic's ViewSim, Meta-Software's HSPICE, and the Tektronix LV500 tester.***

TABLE OF CONTENTS

1. INTRODUCTION

Simulation and test for integrated circuits is generated with the conversion library. The designer need only generate an annotated software emulation of the circuit. Emulation is the first task in a development effort. The conversion library is integrated into this code. When the emulation is run, simulation and test patterns are a byproduct.

Other approaches to the generation of simulation and test stimuli convert from on stimuli to another under software control. Often this is accomplished through an intermediate database format, such as EDIF (Electronic Data Interchange Format) [1], and various software routines to convert stimuli files into EDIF and from EDIF. These approaches characteristically loose information, and thus accuracy, with each conversion.

Simulation and test are functionally similar. The common purpose of any level of simulation and test is to stimulate a circuit and verify the response. The functional sequence (**S**timulus, **O**peration, **a**nd **R**esponse, or **SOAR**) is common to all emulation, simulation, and test.

The remainder of this section is a more complete definition of **SOAR.** The second section describes the conversion library and conversion methods. The third section describes signals and the method used to define timing and voltage levels. The fourth, fifth and sixth sections describe idiosyncrasies of Viewlogic, HSPICE, and Tektronix LV500 conversions. The appendices include the BNF form for definition files and some sample routines.

1.1. SOAR and Conversion

Any software or hardware algorithm illustrates the SOAR principle. Consider the C code below.

$$a = 5;$$

In this simple example, the stimulus applied is the number 5, the operation is assignment, and the result is that the contents of the variable a will be

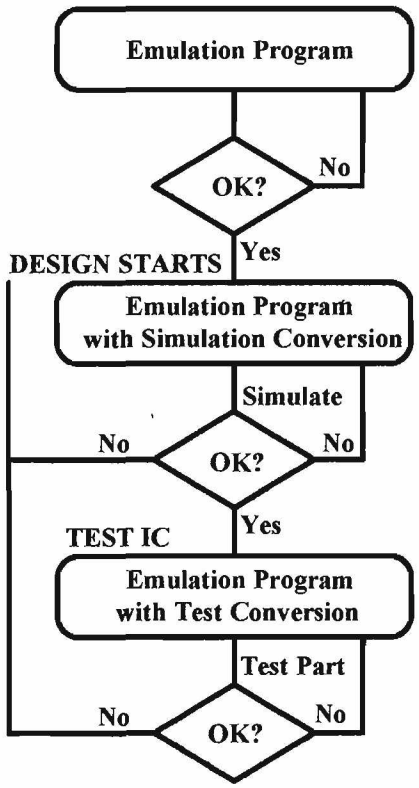


Figure 1: Basic Development Flow

5. The complete behavior of the line has three distinct characteristics, stimulating, operating, and a responding.

Emulation programs capture the functional behavior of integrated circuits. This program is then annotated with calls to the conversion library. The emulation process models the functions of the circuit in an environment. This information is passed to the conversion library. The conversion library translates the state of the emulation to SOAR events in the proper output format: ViewSim, HSPICE, or Tektronix.

The SOAR conversion is the capture of environmental stimulus and integrated circuit response from the emulation, and the translation to the target output format. With a few added constraints to insure the emulation behaves in a manner similar to the target circuit on a cycle-by-cycle basis, the program will capture the functional and cyclical circuit behavior. Figure 1 displays the development flow for the designer.

The annotation of conversion library calls to an emulation program will create gate-level

command files for Viewlogic, a stimulus deck for a HSPICE simulation, or a test program for the Tektronix LV500 tester (see the sample programs in Section 9.

2. CONVERSION

Emulating a circuit with links and calls to the conversion library in the emulation code generates the SOAR events for a simulator or tester. Calls to initialize the conversion, instantiate a SOAR event (a cycle), and complete the conversion are embedded in the emulator. All conversions require a file to define the vectors (see Section 2.2.1). Conversion to test output requires a file to define the load board connections to the circuit (see Section 2.2.2). Optionally, the designer may override the default voltage and timing settings with appropriate files (see Section 3).

The three possible outputs created by the conversion are a Viewlogic simulation command file, a HSPICE stimulus file, or a test program for the Tektronix LV500 tester. Modification of only one constant in the designer's source file, recompilation, and then re-execution of the program will retarget the identical set of stimulus and checks to one of the other target systems.

Default timing and voltage settings for 3.3 and 5 volt CMOS circuits are defined in the conversion library. Two sets of default settings for each circuit type are used, one for simulation and one for test. Table 1 lists the default values for the conversions.

2.1. The Library

The conversion library is linked to the user program during compilation. Currently, Sun platforms running SunOS, Sun platforms running Solaris and Hewlett-Packard platforms running HPUX are supported. The compiler used to generate the library is gcc, and the results are archived into the libraries SOAR.a and SOARD.a. SOARD.a is a debug version of SOAR.a used to analyze library bugs, discussed later.

The library is used as an interface between a generic emulation program and output of specific files for simulation and test (figure2). Definition files are processed directly by the conversion library. Execution errors are reported to the console.

Table 1: Default Time and Voltage Values for Process Selections

Timing	Symbol	SCMOS	SCMOS_TEST	HP14B	HP14B_TEST
Period	Tper	100ns	100ns	6ns	20ns
Clock Delay	Tcd	10ns	10ns	1ns	2ns
Clock Width	Tcw	50ns	50ns	3ns	10ns
Input Setup	Ts	9ns	**5ns**	1ns	**2ns**
Input Hold	Th	10ns	**5ns**	1ns	**1ns**
Output Delay	Td	70ns	70ns	4ns	**4ns**
Output Width	Tdw	10ns	10ns	1ns	8ns
Input Rise	Tr	500ps	500ps	100ps	100ps
Input Fall	Tf	500ps	500ps	100ps	100ps
Voltages					
Power	Vdd	5.0v	5.0v	3.3v	3.3v
Current Limit	Idd	1.0a	1.0a	2.0a	2.0a
Input High	Vih	4.0v	4.0v	3.1v	**3.1v**
Input Low	Vil	0.4v	0.5v	0.2v	**0.5v**
Output High	Voh	2.0v	2.0v	3.1v	1.6v
Output Low	Vol	0.8v	2.0v	0.2v	1.6v

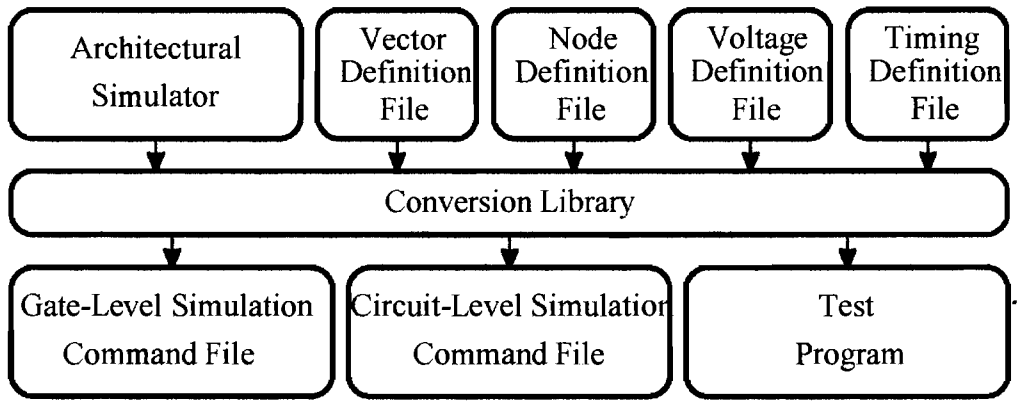


Figure 2: Program Flow of the Conversion Library

2.2. Vector, Node and Pin Definition

Nodes are all electrical connection points in a circuit. For the purpose of conversion, only nodes identified by the designer can be considered. The designer defines the nodes in the vector definition file. The vectors are then included in the linked list data structure in the emulator code.

The net lists which might be simulated are the Viewlogic "vsm" net list format, the Meta HSPICE "sp" net list format, and the actual circuit. Only those nodes specifically named by the designer are potentially available in all three formats. However, internal nodes are not allowed for test equipment. A physical wire cannot be connected to internal nodes.

A group of nodes identified by another name is a vector. The alias name is the vector name. Vector and node names are defined in the vector definition file. Each node must be a member of a vector. Vector names and node names can be identical. Vectors may contain from 1 to 32 nodes.

Vectors are then reported in all but the HSPICE format. HSPICE uses a node representation in the output while the convenience of vectors is maintained in the emulation software.

Vector types can be INPUT, CLOCK, INODE, OUTPUT, BIDIRECTIONAL, POWER, or GROUND. An INODE is a vector of internal nodes. The conversion treats these as outputs except that they can not output to a test program.

2.2.1. Vector Definition File

The vector definition file further defines the vector type, radix, nodes assigned to the vector, and optionally the controlling node and output value for BIDIRECTIONAL vectors. The file is named <fname>.vec, resides in the directory where the conversion is running, and must include all vectors required by the conversion plus POWER and GROUND type vectors. An example file is shown in Figure 3 and the BNF is found in Section 7.

A vector is assigned nodes is sequence. That is the first node in the node list is the left bit in a field (see Figure 4) and the last node is always bit 0. A maximum of 32 nodes may be assigned to the vector. Each line of the definition file is limited to 132 characters.

Node names can be simple, vectored, or elements of vectors. Simple node names begin with an alphabetic character followed by up to 31 alphabetic, numeric, or underscore characters. Vectored nodes follow the rules of simple vectors with the suffix "[n:m]" where n and m are integers either increasing or decreasing. The index for vectored elements is always 1. Elements of vectors are formatted as simple vectors with the suffix "*n*" where n is the integer index for the node. This means "node5" is not the same as "node*5*" and node[5] is illegal.

```
#
# Comments go here; They can not appear in a definition line.
# Power vectors can be defined with or without node names. A node name the same as
#   the vector name is generated when the node names are not specified.
MYPOWERVECTOR      POWER
MYOTHPOWER         POWER      DIRTYPOWER
# Ground vectors and power vectors are defined using the same semantics.
MYGROUNDVECTOR     GROUND
MYOTHGROUND        GROUND     DIRTYGROUND
# Input vectors include a radix and must include a node list. Node names can be identical to vector names.
# Legal radii are binary (BIN), decimal (DEC), and hexadecimal (HEX).
MYINPUT    INPUT   BIN  MYINPUT ANOTHERNODEINPUT
# Files ARE NOT case sensitive in names ONLY.
Myinput2       INPUT   DEC  ADECIMALNODE BDEC CDEC DDEC
# Nodes can be specified as individual members of a vector
myinput3       INPUT   HEX  NODE*0* NODE*1* NODE*2*
# or nodes can be specified as a vector
myinput4       INPUT   HEX  NODE[0:2]
# or as a combination
myinput5       INPUT   HEX  NODE[0:2] NODE*5*
# Clock nodes do not use the radix. They are always binary radix.
Myclock        CLOCK          CLK_ONE CLOCK_2
# Outputs and inodes are treated identically. They require the radix and the node name list.
myoutput       OUTPUT   BIN   out[5:0]
myinode        INODE    BIN   inode[7:0] another_inode_name
# BIDIRECTIONAL type vectors also require a controlling node (of INPUT, INODE, or OUTPUT type)
#   and the value on the control node that causes the BIDIRECTIONAL vector to be in output mode.
# The following line creates a bi-directional vector with 32 nodes, formatted in hexadecimal, which
#   outputs data when the node "Control_Node_Name" is defined and holds the value 1. The library
#   treats the BIDIRECTIONAL vector as masked if the controlling node is not defined.
Mybidirectional   BIDIRECTIONAL    HEX   Control_Node_Name  1  bidi[31:0]
```

Figure 3: Sample Vector Definition File

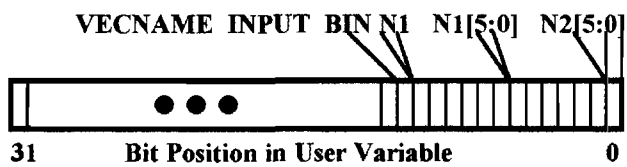


Figure 4: Map of Bit Position to Nodes

A special case of node assignment is found in the POWER and GROUND vectors. The node list may be optionally omitted. When omitted the vector becomes a vector of one node named the same name as the vector.

Radix definition is required for INPUT, OUTPUT, INODE, and BIDIRECTIONAL vectors. Binary (BIN), decimal (DEC), and hexadecimal radii are allowed. Support for the different radii varies in the target systems. If the target does not support the designer's choice, an alternate is selected. The CLOCK vectors default to binary radix. Radix definition does not apply to POWER and GROUND vectors.

BIDIRECTIONAL vectors require the name of the controlling node and the control value to determine the state, input or output, of the BIDIRECTIONAL vector. The controlling node

```
# Comments start with a "#"
# all command lines have the form <node_name> <pin_number> <sector>.<channel>
powernode    40  x.x
groundnode   20  x.x
signalnode1  5   4.3
signalnode2  6   a.b
signalnode3  K5  c.d
```

Figure 5: Sample Pin Definition File

is first and foremost a node, not a vector. Second, the controlling node must be of INPUT, OUTPUT or INODE type. The control value must be either "0" or "1".

2.2.2. Pin Definition File

The pin definition file maps nodes to package pin numbers and Tektronix LV500 pin electronics channels. This static definition representing physical hardware allows logical reference to physical connections. The complete BNF for this file can be found in Section 7. A sample file is shown in Figure 5. The file is named <fname>.pin and is located in the current directory.

The pin number, or name, can be any string of alphabetic and numeric characters, limited to 32 total characters. This is a required comment for the Tektronix LV500 tester. It is useful during circuit analysis and debug.

The tester channel and sector are single hexadecimal digits that define the 16 sectors of 16 channels each on the test head. These are wired to the integrated circuit through the load board. Thus, the sector and channel assignment is derived from the load board connections. A special sector.channel, x.x, is reserved to denote the hard-wired power and ground pins on the load board.

2.3. The User Interface

The programmer links to the conversion library in three steps. First, definition files are created to define vectors, nodes, pins, timings, and voltages. Second, data structures, static or dynamic, are created to define the vectors within the C program. Third, the three conversion library procedure calls are integrated into the C program. The static tasks, definition file and data structure creation, define the integrated static conversion parameters.

2.3.1. Data Structures

The primary data structure used to link the designer's program to the conversion library, Figure 6, is a simple linked list containing the vector's name, the address of the unsigned long vector value, the definition field, and a link to the next data structure. Variable length vector lists are supported by the linked list structure. The only limit to the list length is available memory on the machine and available patience during processing.

The vector name field is actually a pointer to a string containing the vector name. The name is used during the initial symbol table generation. All vectors in the vector definition file must appear in the linked list and vice versa. A vector name may contain up to 32 characters.

The vector value address points to the data storage location. The conversion library monitors this location to set values and schedule changes in values as the emulation executes. Viewlogic and Tektonix conversions schedule vector value changes to generate the output file. HSPICE conversions decompose the vector into nodes at each SOAR event. The HSPICE schedule is defined by individual node.

```
typedef struct vector_link
{
  char *vector_name;              /* Names of the Vector */
  unsigned long *value;           /* Address of the Value */
  BOOLEAN defined;                /* Value OK or undefined */
  struct vector_link *next;       /* Next Link */
} VECTOR_LINK;

void Init_SOAR ( char *file_root, VECTOR_LINK *link,
        CONVERSION_TYPE con, DESTINATION_TYPE dest );
void SOAR ( char *cstring );
BOOLEAN Finish_SOAR ( void );
```

Figure 6: Data Structures and Procedure Prototypes

The vector definition field is a boolean value. If the value of a vector is unknown at any point in the emulation, this value is set false. Table 2 lists the possible combinations of truth and vector type with the resulting conversion action.

BIDIRECTIONAL vectors require an output state defined by the controlling node's value. The method to set the controlling node is discussed in the previous section on the vector definition file.

Table 2: Input/Output Response to Vector Definition

	Vector		Control		Result		
Vector Type	**Defined**	**Value**	**Defined**	**Value**	**HSPICE**	**Viewlogic**	**Tektronix**
CLOCK	TRUE	0	-	-	RO Format	RO Format(1)	RO Format(1)
CLOCK	TRUE	1	-	-	RZ Format	RZ Format(2)	RZ Format(2)
CLOCK	FALSE	either	-	-	Coerced(3)	Coerced(3)	ERROR
INPUT	TRUE	0	-	-	0 Driven	0 Driven	0 Driven
INPUT	TRUE	1	-	-	1 Driven	1 Driven	1 Driven
INPUT	FALSE	either	-	-	Coerced(3)	Coerced(4)	ERROR
OUTPUT	TRUE	either	-	-	-	Checked	Compare
OUTPUT	FALSE	either	-	-	-	Not Checked	Masked
INODE	TRUE	either	-	-	-	Checked	Compare
INODE	FALSE	either	-	-	-	Not Checked	Masked
BIDIRECTIONAL	TRUE	0	TRUE	IN	0 Driven	0 Driven	0 Driven
BIDIRECTIONAL	TRUE	1	TRUE	IN	1 Driven	1 Driven	1 Driven
BIDIRECTIONAL	TRUE	either	TRUE	OUT	-	Checked	Compare
BIDIRECTIONAL	FALSE	either	TRUE	OUT	-	Not Checked	Masked
BIDIRECTIONAL	-	-	FALSE	either	-	Not Checked	Masked

NOTES:

1: Remains RO format; Changing the logic value or setting false removes the clock for a cycle.
2: Remains RZ format; Changing the logic value or setting false removes the clock for a cycle.
3: This signal is coerced throughout the vector set.
4: The signal is released (X input) after the first cycle.

2.3.2. Routines

The three procedures linking the user software to the conversion library are Init_SOAR, SOAR, and Finish_SOAR as shown in Figure 7. Init_SOAR defines the symbol tables from the vector definition file and the vector links to the user program. It also reads the timing and voltage definition files to define the resources required. SOAR executes one SOAR event step, or period. Finish_SOAR outputs the target file and performs various housekeeping tasks.

The call to Init_SOAR initializes the conversion library by identifying data links for all vectors except POWER and GROUND types. The initialization call then reads the vector definition file, matching the linked vector list to the defined vectors. This establishes the node and vector symbol tables. These tables are static for the remainder of the conversion except for the old and new value fields. The only opportunity for the conversion library to change a user data structure occurs during initialization. If an input begins undefined, the routine coerces it to defined and notifies the user.

Each SOAR event is created by the SOAR procedure call. The single argument is an optional comment for the SOAR event. Comments are included in the output. A list of comments with cycle number is created for HSPICE. ViewSim and the LV500 files include comments at the appropriate time point in the output file. A zero length string inserts no comment.

The Finish_SOAR procedure performs three tasks. It checks for inconsistency errors in the conversion. It converts the schedule structure in memory to the SOAR event sequence. Finally, it frees the dynamic variables used by the library. This routine has no parameters.

2.3.3. Bug Identification and Reporting

To assist in identifying bugs, a debug version of the library is available, SOARD.a. If an error is believed to be located in the conversion library, recompile your emulation with this version of the library. When you run your program, a log file, <fname>.log will be created. This library is larger and executes slower than the standard library. The log file created can be large as it is a trace of the library execution.

E-mail the author at masoares@cs.utah.edu to report errors. Include the location of the unprotected source files and log file. Also, include the machine name and operating system used.

3. SIGNAL DESCRIPTION

Signals are the voltage values that are assigned or sensed at nodes in a circuit. A complete sequence of electrical values applied and checked on the nodes constitutes a simulation or test. To properly generate signal traces or sequences, the conversion library must know the node type (input, output, etc.). The point in time to either apply or check a node must be defined. Circuit simulation and test also need to the applied voltage defined.

The applied or sensed logic is defined by the current logic value from the emulation link. At each SOAR event (or call), the values of vectors are compared to the previous values of vectors and all changes are scheduled. Timing and voltage values are defined by the process default values and values set by the timing definition and voltage definition files.

3.1. Node and Vector Types

Vectors can be defined as INPUT, CLOCK, OUTPUT, INODE, BIDIRECTIONAL, POWER, or GROUND type. The vector type defines the its use during conversion and subsequent simulation or test. INPUT and CLOCK types are signals applied to the circuit. OUTPUT and INODE types are checked for accurate responses. BIDIRECTIONAL vectors are either INPUT or OUTPUT in nature, dynamically changing. POWER and GROUND types supply the DC power for the circuit.

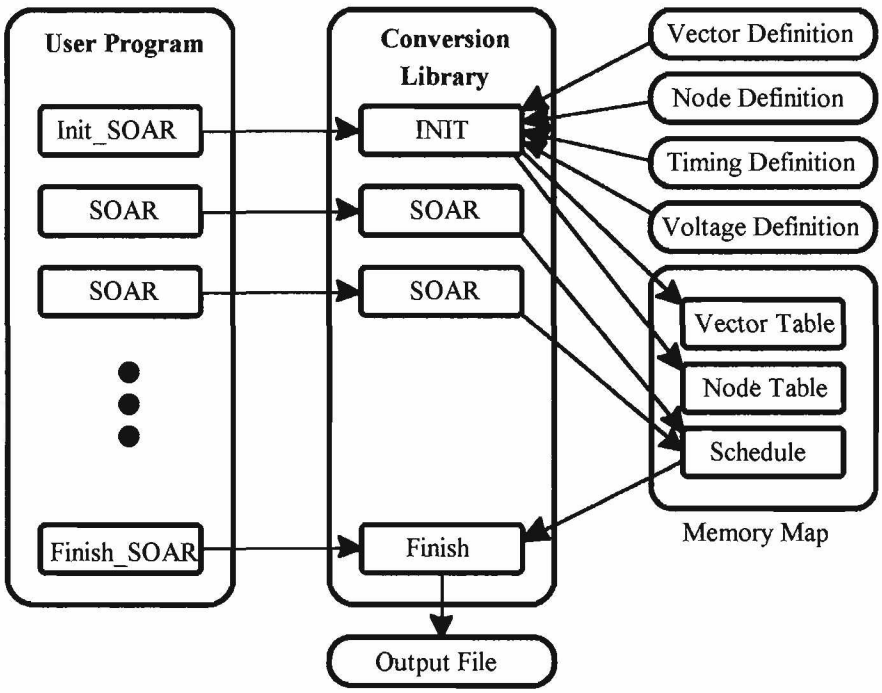


Figure 7: Interaction between User Software and the Conversion Library

3.2. Formatting Signals

All signals are referenced to the clock signal. The period is the heartbeat of SOAR events. This period does not have to represent a clock signal, but must be defined. All timings, default and definition file, are relative to the period. CLOCK signals are referenced from the beginning of the period. All other signals are referenced from the first clock edge. The special vector name "DUMMY_CLOCK" is reserved for a CLOCK vector that is not representing a circuit node. The CLOCK, whether dummy or real, defines the period time and provides the reference edge for all other signals. If no CLOCK vector is specified for a circuit, the default values for period and clock delay are used.

Figure 8 shows the timing relationships. The length of a SOAR event equals the period of a clock, t_{per}. The first edge, the active edge, of a clock is delayed into the event by the clock delay, t_{cd}. All other edges are placed in reference to the active clock edge. The dummy clock can substitute for this signal. All timing events must occur during the SOAR event. Edges occurring outside the SOAR event, before or after, are not allowed. The conversion type further restricts edge placements. These restrictions are discussed in the conversion specific sections.

CLOCK vectors always use return-to-zero (RZ) or return-to-one (RO) format (RZ is shown in Figure 8). The format is determined by the CLOCK vector value at initialization. A value of '1' set RZ format. A value of '0' sets RO. Each CLOCK node in the vector is individually set to its format by specifying the binary digit value. The use of multiple clocks, and in particular, multiple clocks with varying frequency or phase relationship, is dependent upon the target system. Conversion specific CLOCK restrictions are specified in later sections.

For simulation, the INPUT type can be formatted as non-return-to-zero (NRZ) or return-to-complement (RTC) as shown in Figure 8. The Tektronix tester will always use the NRZ format. Default for INPUT vectors in simulation is the RTC format. To set the NRZ format, the designer must use the timing definition file. Any input hold time equal to the negative of the setup time, i.e. simultaneous edges setup and hold, will switch the signal to NRZ format.

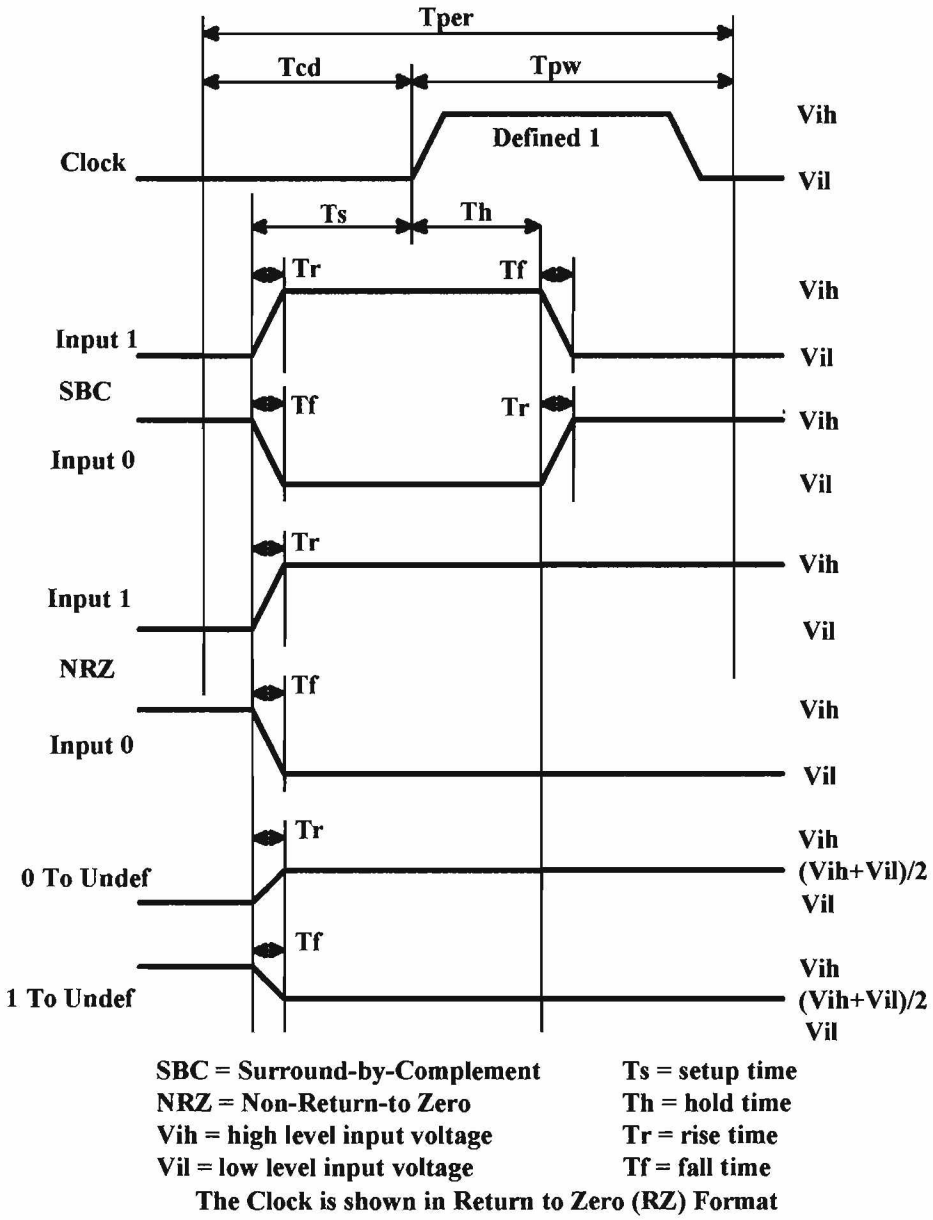


Figure 8: Timing and Formatting

The rise and fall times, t_r and t_f, are used in HSPICE conversion to create true waveforms for CLOCK, INPUT, and BIDIRECTIONAL (when in INPUT mode) vectors. Gate-level simulation has no provision for rise and fall times. The Tektronix LV500 uses hardware-fixed rise and fall times. The voltage slew rate for the LV500 signals is approximately two volts per nanosecond.

The input voltage reference levels, V_{IH} and V_{IL}, are required for circuit simulation and test These levels correspond to logic one and zero respectively. There are resource limits on the LV500 tester discussed in Section 6.

The output voltage levels, V_{OH} and V_{OL}, are used only by the Tektronix LV500. The two levels are provided to be generic with all test equipment. The Tektronix only uses VOH as the hardware compare circuit only has one voltage rail.

OUTPUT and BIDIRECTIONAL vectors are compared to logical data using one of three

methods as shown in Figure 9. The Tektronix LV500 compares OUTPUT vectors to the logical data for the entire window between t_d and t_{dh}. BIDIRECTIONAL vectors are only compared to data at the t_d edge. ViewSim compares the logical output to the data on both the t_d and t_{dh} edges. Comparison to output data is not available with circuit simulation.

The POWER vector is used to define the power supply nodes. HPSICE requires the supply voltage and the Tektronix LV500 requires both the voltage and current limit. Only one supply voltage may be specified for the tester, and this voltage can be applied to multiple nodes. There are restrictions placed on the power supplies by both the HSPICE and LV500 conversions. Both require all reference values be greater than or equal to 0 volts and less than or equal to the highest power supply setting. The Tektronix further restricts the power supply range to 6 volts maximum.

3.3. Timing Specification

Default times are used unless different timings are assigned in the time definition file. Default timings can be selected from a set of four during the initialization call to the library. If NRZ timing is required, the timing definition file must be used. The defaults are shown in Table 1.

The timing definition file can be used to override the default timings. An example is shown in Figure 10. This file must be resident in the directory where the conversion is executing ad be named <fname>.time. Nodes or vectors may be specified. However, Viewlogic and the tester only use vector values. Each command line of the timing file contains a sequence of times. Each field's context determined by the vector type.

If NRZ format is required, the input hold time, t_h, must equal the negative of the setup time, t_s. When the start and stop edges of the data coincide, the conversion library asserts NRZ format instead of a 0 length pulse.

3.4. Voltage Specification

Voltage is set, like timing, by the default values of a process, see Table 1. Individual voltage values may be changed by the voltage definition file contents. An example voltage definition file is shown in Figure 11. This definition file must reside in the same directory as the executing conversion and have the name <fname>.volt.

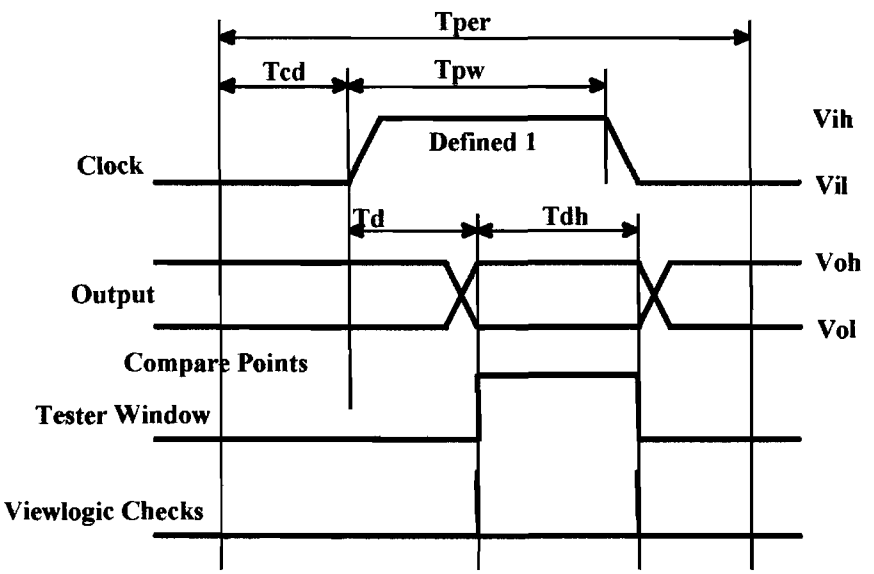


Figure 9: Compare Timing

```
# Comment lines begin with a "#". They are not allowed on a command line.
# The following line shows the definition of a clock vector or node.
# The clock sets the period to 45ns with a 5ns delay and 22.5ns width
# The clock will rise in 100ps and fall in 200ps
#Clock          Tper    Tcd     Tpw     Tr      Tf
CLOCKNODE       45e-9   5e-9    22.5e-9 1e-10   2e-10
# The following line illustrates an input definition given the line above for a vector or node
# The setup time is 3ns which means it occurs 2ns into the period
# The hold time is 10ns which means it occurs 15ns into the period
# The rise and fall times are 220ps and 230ps respectively
#Input          Ts      Th      Tr      Tf
INPUTVECTOR     3e-9    10e-9   2.2e-10 2.3e-10
# The following lines illustrate a inode or output timing declaration.
# The delay is 8ns or 13ns into the period
# The width is 6ns or 19ns into the period
#Output or inode Td      Tdh
OUTPUTNODE      8e-9    6e-9
INODEVECTOR     8e-9    6e-9
# The following line illustrates a bidirectional node or vector.
# The setup time is 1ns or 4ns into the period and the hold time is 2ns or 7ns into the period.
# The rise and fall times are 90 and 110ps respectively.
# The output delay is 9ns or 14ns into the period
# The output width is 22ns or 36ns into the period
# Bidirectional  Ts      Th      Tr      Tf      Td      Tdh
BIDINODE        1e-9    2e-9    9e-11   1.1e-10 9e-9    22e-9
# The following illustrates an input which is set to NRZ format with a 2ns setup time
# The rise and fall times are both 100ps
# Input          Ts      Th      Tr      Tf
NRZINPUT        2e-9    -2e-9   1e-10   1e-10
```

Figure 10: Timing Definition File

```
# Comment lines begin with a "#". They are not allowed on a command line.
# The following line shows the definition of a clock or input vector or node.
# The power supply, Vaa, is set to 4.8 volts at a 1.1 amp current limit.
# Power          Voltage Current
Vaa             4.8     1.1
# The input high level is set to 2.8 volts for the clock and 2.3 volts for the input
# The input low level is set to 0.2 volts for the clock and 0.4 volts for the input
#Clock or input  Vih     Vil
CLOCKNODE       2.8     0.2
INPUTVECT       2.3     0.4
# The following lines illustrate a output voltage declaration.
# The output compare voltage is 2.0 and 0.8
#Output         Voh     Vol
OUTPUTNODE      2.0     0.8
# The following line illustrates a bidirectional node or vector.
# The voltages are a combination of input and output voltages
# Bidirectional  Vih     Vil     Voh     Vol
BIDINODE        2.4     0.3     1.6     1.6
```

Figure 11: Voltage Definition File

4. VIEWLOGIC CONVERSION

Viewlogic conversion outputs the command file for a ViewSim gate-level simulation [6]. This file is easily invoked in ViewSim by executing the command file. The command file generated by the conversion software will (re)load the net list, set up vectors, establish waveform output, watch output, log output, trace output and the execute the simulation. A file, <fname>,log contains outputs not reported to the screen. <fname>.wfm contains the waveform. The waveform can be viewed in ViewTrace.

To execute any conversion to Viewlogic, the designer must invoke the conversion with the VIEWLOGIC switch as shown in Figure 12. The vector definition file must be present in the current directory. Optionally the timing definition file may be used. The pin definition and voltage definition files are not used by this conversion.

Use of the ViewSim assignment method for a clock signal allows the use of multiple clocks if and only if the frequencies are an integer multiples of one another. Only one period is allowed. The one period is selected by using the first CLOCK timing encountered. The methods used to obtain other clocks are discussed in Section 4.1.

ViewSim can "float" inputs during a simulation. ViewSim sets the node to a logic "X," or undefined The conversion routine will issue a warning and set the input definition field to true on the first cycle only. The designer must set the defined portion of the data structure to false to set an input to "X."

BIDIRECTIONAL vectors are converted to values dependant on the current state of the controlling node. If the node changes during a SOAR event, the input/output state of the BIDIRECTIONAL vector will change according to a number of factors as shown in Table 3. In general the program will err on the conservative side, releasing the value early and setting it late.

```
Init_SOAR( "my_file_name", &my_linked_list, SCMOS, VIEWLOGIC );
```

Figure 12: Invoking a Viewlogic Conversion

Table 3: BIDIRECTIONAL Switching in Viewlogic

CONTROL			BIDIRECTIONAL			
TYPE	FROM	TO	FROM	TO	CHANGE	WHEN
INPUT	U, X	D, in	D, value	D, value	drive value	Ts(control)
OUTPUT	U, X	D, in	D, value	D, value	drive value	Td(control)
INODE	U, X	D, in	D, value	D, value	drive value	Td(control)
INPUT	D, out	D, in	D, value	D, value	drive value	Ts(control)
OUTPUT	D, out	D, in	D, value	D, value	drive value	Td(control)
INODE	D, out	D, in	D, value	D, value	drive value	Td(control)
ANY	U, X	D, out	D, value	D, value	release	beginning
ANY	U, X	D, out	D, value	D, value	check value	Td(bidirectional)
ANY	D, in	D, out	D, value	D, value	release	beginning of event
ANY	D, in	D, out	D, value	D, value	check value	Td(bidirectional)
ANY	D, in	U, X	D, value	D, value	release	beginning of event
ANY	D, in	D, in	D, value	D, new	drive new	Ts(bidirectional)
ANY	D, out	D, out	D, value	D, new	check new	Td(bidirectional)

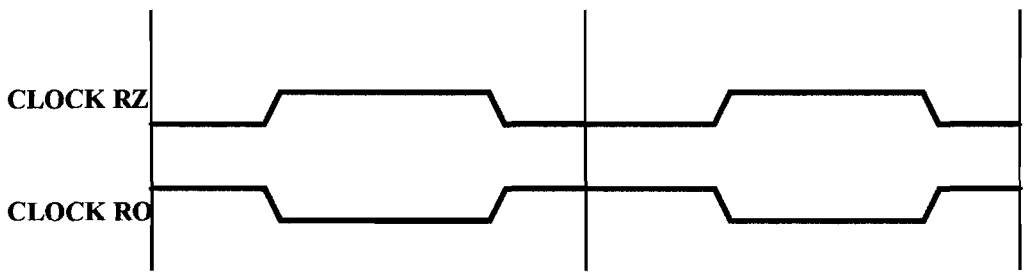
4.1. Clock Restrictions

Multiple clock signals can be manipulated to a large degree with Viewlogic. The only restrictions are that multiple clock frequencies must be integer power-of-two multiples of one another and the designer may be required to declare some clocks as inputs as shown in Figure 13.

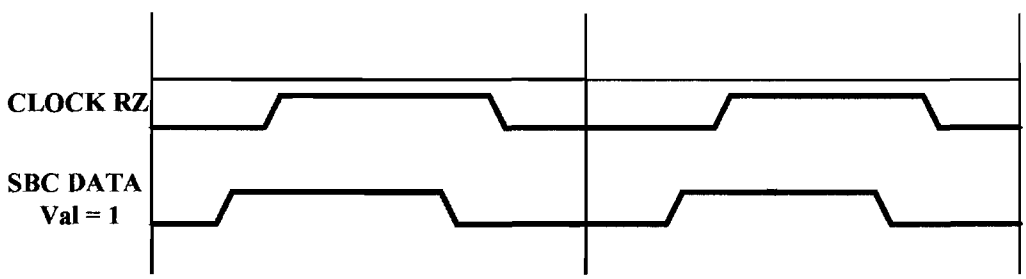
In the first case of Figure 13, two clocks of opposite polarity are created when the initial data value is differs. By setting the initial value, the designer controls whether the clock is RZ or RO format. This mode is limited to clocks with simultaneous edges, i.e. the delay and width are identical.

After the initial value for a clock is used to determine the format, RZ or RO, the logic value is used to enable or disable the clock. When a SOAR call is make with the clock logic value equal to the initial value, a clock pulse is issued. However, if the value is the complement of the initial value, no clock pulse is issued. However, this clock can be omitted from a SOAR event by complementing the value. The clock will then resume when the value is returned to the initial value.

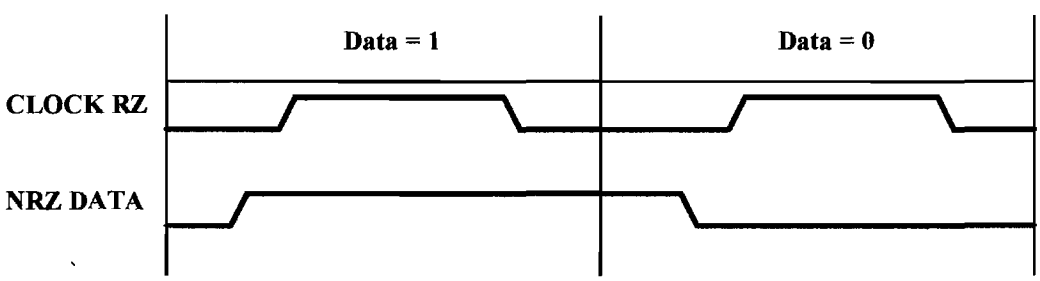
In the second case, the phase relationship can be manipulated when one of the clocks is defined as an INPUT node with SBC format. The phase relationship is then declared through the setup and hold times for the INPUT type clock.. The INPUT type clock cannot be disabled.



(a) Clock, Return-to-Zero(1 data) and Return-to-One(0 data)



(b) Clock and SBC Data as Clock, Phase Shift



(c) Clock and NRZ 2x Clock, Multiples with Phase Shift

Figure 13: Clock Generation

The third case includes all clocks with periods that are a power of two multiple of the original clock. The second clock is defined as an INPUT type with NRZ format. The data value is then modified as the emulation proceeds. This method has the advantages of modifying the phase relationship with the setup time and allowing suspension of the clock.

5. HSPICE CONVERSION

HSPICE conversion targets a stimulus file for an HSPICE simulation. This file is added to the spice deck, i.e. the transistor net list. The resulting file is then simulated with HSPICE[2][3][4]. The stimulus file, called "<fname>.stimulus," includes the appropriate library of transistor models, standard global values (POWER and GROUND nodes), power supplies, piece-wise linear supplies for the INPUT nodes, and a special circuit controlled by the control node for BIDIRECTIONAL vectors.

HSPICE conversion requires the vector definition file. HSPICE will optionally read and use the optional timing definition and voltage definition files. All files used must reside in the current directory. The designer need only compile and execute the program with HSPICE selected in the Init_SOAR invocation as shown in Figure 14.

```
Init_SOAR ( "my_file_name", &my_linked_list, SCMOS, HSPICE );
```

Figure 14: Invoking a HSPICE Conversion

Clock signals have no restrictions on placement or frequency in HSPICE conversion. The conversion software converts these signals into PULSE statements. The PULSE supply repeats a waveform for the duration of transient analysis. More information is available in [2. The SOAR period is defined by the first CLOCK period defined. Further, all the setup and delay edges are calculated from this first clock. All timing edges must be placed inside the SOAR period.

All INPUT signal waveforms have NRZ or SBC format. HSPICE is unique in that the assignment is made to each individual node, not vectors The conversion library makes this distinction transparent to the user.

No comparison for correct OUTPUT, BIDIRECTIONAL, and INODE values is made. The BIDIRECTIONAL nodes are connected to a piece-wise linear resistor. High impedance is set while the BIDIRECTIONAL node is in output mode. Low impedance to a piece-wise linear source is connected during input mode. The value of the resistor is set by the control node value.

Input reference values, V_{IH} and V_{IL}, are used to set the voltage applied for a logical zero or one. Output reference voltages are not used. The power supply voltage setting is connected to the POWER nodes. There is no limit on the number of power supplies used. There is also no limit on the voltage value except that it must be positive.

Inputs are always initially coerced. If an INPUT node is undefined during initialization, the node is set to the value, the defined field in the vector link is set defined, and a warning is issued. At any other time during conversion, the input is set midway between V_{IH} and V_{IL} when the calling routine sets the INPUT node undefined.

6. TEKTRONIX LV500 TESTER

Tektronix LV500 conversion generates a "msa" file to be used as a test program [5]. This file is transferred to the tester by ftp. Compilation from the "msa" format to the test binary is performed on the LV500. The file, <fname.msa>, is a complete test program. All levels, timings, assignments and vectors are defined.

Conversion to the Tektronix "msa" format is defined by the TEKTRONIX switch in the

initialization call to the library as shown in Figure 15. The vector definition and pin definition files are required and must be present in the current directory. Optionally the timing definition and voltage definition files may be used to alter the defaults.

The LV500 tester has physical restrictions on allowed tests. For example, the tester cannot probe internal nodes. INODE type vectors are omitted from the output file and a warning is issued. These nodes can still be used to change direction for BIDIRECTIONAL vectors, but they do not appear in the output file.

Another test restriction is the absolute switching bitm for all BIDIRECTIONAL vectors. These vectors always change mode (input to output, or output to input) at the SOAR period boundary. The user should use simulation to examine the effect of this boundary switch prior to test.

All INPUT vectors must be defined. The test equipment cannot drive a third level during test. Even if it could, what is the voltage of an unknown? Consequently, undefined inputs are an error.

6.1. Power Supplies and References

The LV500 has only one useful power supply and two sets of reference voltages. The power supply is connected to the appropriate circuit pins by physical wires on the load board. The two sets of references are assigned to pins under program control, but they are assigned in groups of eight pins. A full discussion of this restriction appears in the *LV500 Operator's Manual* [5].

The power supply voltage range is 0.50 volts to 6.00 volts with a 50 millivolt resolution. The power supply has a programmable current limit. The limit is programmable from 0.1 amps to 2.0 amps with a resolution of 0.1amps.

The two reference sets are programmable. Each reference set includes a value for V_{IH}, V_{IL}, and V_O. Only one output compare level is available on the LV500. The conversion library software uses the V_{OH} level for the output reference. Table 4 shows the range and limits for these supplies.

6.2. SOAR Period Timing

The tester has two timing ranges, shown in table 5. One timing range must be used for all timing edges. The conversion library chooses the timing range by selection the best resolution possible to the defined period length. A LV500 restriction on timing edges is that no edge may be placed during the "dead time." The dead time is the last "n" nanoseconds of a period.

Each SOAR event constitutes one LV500 pattern vector being applied and checked in the period. Up to 64,000 vectors, decimal, not binary, may be applied. Each comment or SOAR event translates to one LV500 vector.

Table 4: Reference Range

Value	Min	Max	Step
V_{IH}	V_{IL} + 0.5	V_{DD}	0.05v
V_{IL}	0.50v	V_{IH} - 0.5	0.05v
V_O	0.50v	V_{DD}	0.05v

Table 5: Timing Ranges

Value	Resolution	Range	Step	Dead Time
Period	High	>= 20ns <= 100ns	2ns	8ns
Period	Low	>= 100ns <= 4096ns	20ns	80ns
Edge	High	>= 0ns <= Period - 8ns	500ps	8ns
Edge	Low	>= 0ns <= Period - 80ns	20ns	80ns
Pulse Width	High	>= 8ns	2ns	20ns
Pulse Width	Low	>= 80ns	20ns	80ns

```
Init_SOAR ( "my_file_name", &my_linked_list, SCMOS, TEKTRONIX );
```

Figure 15: Invoking a Tektronix Conversion

6.3. Clock Restrictions

The CLOCK vectors on the tester are restricted to RZ and RO format with the same delay and width values for all clocks. Like the ViewSim "tricks" (Section 4.1), different phased and multi-cycle CLOCK vectors can be created using the INPUT type. Since INPUT vector timings are very restricted, a multi-cycle CLOCK vector will be also. The next section describes the restrictions.

6.4. Input Creation

INPUT vectors are be driven by the LV500 at all times during the SOAR event execution. Any undefined INPUT value will trigger an error. The INPUT vectors are all at either V_{IH} or V_{IL} levels. Only one transition per period is allowed (NRZ format). All INPUT vectors use the same timing generator and are thus changed simultaneously.

The LV500 includes resources that can change the INPUT restrictions. These other resources remove the restrictions mentioned above, but add others. For example, if the designer requires two different INPUT timing edges, the LV500 will supply them if the load board is wired properly. The Tektronix *LV500 Operator's Manual* [5] describes the resources in more detail The conversion library does not support all possible test options.

6.5. Output Checks

The circuit output is compared to expected data during the entire time from the t_d to the t_{dh} timing edges. The LV500 uses a gated latch to capture output glitches as well as functional errors. Any deviation from expected data will latch an error for that SOAR period. Internal nodes cannot be checked.

All OUTPUT vectors share one timing generator for comparison. Like the INPUT vectors, there are specific restricted variations on the LV500 found in the Tektronix *LV500 Operator's Manual* [5].

6.6. Bidirectional Processing

BIDIRECTIONAL vectors are connected to a fourth timing generator. The CLOCK vectors take one generator, the INPUT vectors a second, and the OUTPUT vectors a third. The BIDIRECTIONAL vectors use the first edge of a timing generator to time inputs and the second edge to time the OUTPUT comparison.

BIDIRECTIONAL vectors in input mode use NRZ format. The first edge of the timing generator determines the switching edge time. Switching modes to input occurs at the period boundary only. This eliminates bus contention assuming the user's software is accurate.

BIDIRECTIONAL vectors in output mode use the trailing edge of the timing generator to make comparisons. The actual circuit data is compared to the expected response at the timing edge only. The resolution of the trailing edge is less than the leading edge. The conversion routine rounds the desired value to the nearest legal value.

The Tektronix LV500 does not support timing generator based IO switching on the pins. Therefore, input/output switching always occurs on the SOAR period boundary. There are resources and formats in the LV500 that can be useful when the approach here is inadequate. See the Tektronix *LV500 Operator's Manual* [5] for complete information.

6.7. Channels and Sectors

The Tektronix LV500 tester contains 256 pin electronics channels. Thus up to 256 signal pins may be connected to the tester. This limit does not include power and ground pins. The cards are addressed as one of sixteen channels in one of sixteen sectors. Card addresses are defined in the required pin definition file.

7. DEFINITION FILE BNF

The BNF form of the four possible definition files is given in Figures 16 through 19. The syntax was created with two design goals. First, the syntax should be simple. Second, the syntax of the various files should be similar.

```
VectorFile ::= { VectorDefs | Comments }

alpha ::= 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' | 'h' | ' i' | 'j' | 'k' | 'l' | 'm' | 'n' | 'o' | 'p' | 'q' | 'r' | 's' | 't' | 'u' |
          'v' | 'w' | 'x' | 'y' | 'z' | 'A' | 'B' | 'C' | 'D' | 'E' | 'F' | 'G' | 'H' | 'I' | 'J' | 'K' | 'L' | 'M' | 'N' |
          'O' | 'P' | 'Q' | 'R' | 'S' | 'T' | 'U' | 'V' | 'W' | 'X' | 'Y' | 'Z'
anychar ::= alpha | numeric | specialchar
Comments ::= '#' { anychar }
ControlValue ::= '0' | '1'
NodeName ::= alpha { alpha | numeric | '_' } [ ( '[' numeric ':' numeric ']' ) | ( '*' numeric '*' ) ]
numeric ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
Radix ::= 'BIN' | 'DEC' | 'HEX'
specialchar ::= ''' | '~' | '!' | '@' | '#' | '$' | '%' | '^' | '&' | '*' | '(' | ')' | '-' | '_' | '=' | '+' | '[' | ']' |
          '{' | '}' | ';' | ':' | ''' | '"' | ',' | '.' | '/' | '<' | '>' | '?'
VectorDefs ::= VectorName VectorTypes
VectorName ::= alpha { alpha | numeric | '_' }
VectorTypes ::= ('POWER' { NodeName } ) |
          ('GROUND' { NodeName } ) |
          ('CLOCK' NodeName { NodeName } ) |
          ('INPUT' Radix NodeName { NodeName } ) |
          ('OUPUT' Radix NodeName { NodeName } ) |
          ( 'BIDIRECTIONAL' Radix NodeName ControlValue NodeName { NodeName } ) |
          ( 'INODE' Radix NodeName { NodeName } )
```

Figure 16: Vector Definition File BNF

```
NodeFile ::= { NodeDefs | Comments }

alpha ::= 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' | 'h' | ' i' | 'j' | 'k' | 'l' | 'm' | 'n' | 'o' | 'p' | 'q' | 'r' | 's' | 't' | 'u' |
          'v' | 'w' | 'x' | 'y' | 'z' | 'A' | 'B' | 'C' | 'D' | 'E' | 'F' | 'G' | 'H' | 'I' | 'J' | 'K' | 'L' | 'M' | 'N' |
          'O' | 'P' | 'Q' | 'R' | 'S' | 'T' | 'U' | 'V' | 'W' | 'X' | 'Y' | 'Z'
anychar ::= alpha | numeric | specialchar
Comments ::= '#' { anychar }
hex ::= numeric | 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'x' | 'A' | 'B' | 'C' | 'D' | 'E' | 'F' | 'X'
NodeDefs ::= NodeName PinNumber SectorChannel
NodeName ::= alpha { alpha | numeric | '_' } [ ( '[' numeric ':' numeric ']' ) | ( '*' numeric '*' ) ]
numeric ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
PinNumber ::= ( alpha | numeric | '_' ) { alpha | numeric | '_' }
SectorChannel ::= hex '.' hex
specialchar ::= ''' | '~' | '!' | '@' | '#' | '$' | '%' | '^' | '&' | '*' | '(' | ')' | '-' | '_' | '=' | '+' | '[' | ']' |
          '{' | '}' | ';' | ':' | ''' | '"' | ',' | '.' | '/' | '<' | '>' | '?'
```

Figure 17: Pin Definition File BNF

```
TimeFile ::= { TimeDefs | Comments }

alpha ::= 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' | 'h' | ' i' | 'j' | 'k' | 'l' | 'm' | 'n' | 'o' | 'p' | 'q' | 'r' | 's' | 't' | 'u' |
          'v' | 'w' | 'x' | 'y' | 'z' | 'A' | 'B' | 'C' | 'D' | 'E' | 'F' | 'G' | 'H' | 'I' | 'J' | 'K' | 'L' | 'M' | 'N' |
          'O' | 'P' | 'Q' | 'R' | 'S' | 'T' | 'U' | 'V' | 'W' | 'X' | 'Y' | 'Z'
anychar ::= alpha | numeric | specialchar
Comments ::= '#' { anychar }
Decimal ::= Integer '.' [ Integer ]
Float ::= Decimal ( 'e' | 'E' ) Integer
Integer ::= [ '-' ] numeric { numeric }
NodeName ::= alpha { alpha | numeric | '_' } [ ( '[' numeric ':' numeric ']' ) | ( '*' numeric '*' ) ]
numeric ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
Parameters ::= Float | Decimal | Integer
specialchar ::= '`' | '~' | '!' | '@' | '#' | '$' | '%' | '^' | '&' | '*' | '(' | ')' | '-' | '_' | '=' | '+' | '[' | ']' |
          '{' | '}' | ';' | ':' | ''' | '"' | ',' | '.' | '/' | '<' | '>' | '?'
TimeDefs ::= NodeName { Parameters }
```

Figure 18: Timing Definition File BNF

```
VoltFile ::= { VoltDefs | Comments }

alpha ::= 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' | 'h' | ' i' | 'j' | 'k' | 'l' | 'm' | 'n' | 'o' | 'p' | 'q' | 'r' | 's' | 't' | 'u' |
          'v' | 'w' | 'x' | 'y' | 'z' | 'A' | 'B' | 'C' | 'D' | 'E' | 'F' | 'G' | 'H' | 'I' | 'J' | 'K' | 'L' | 'M' | 'N' |
          'O' | 'P' | 'Q' | 'R' | 'S' | 'T' | 'U' | 'V' | 'W' | 'X' | 'Y' | 'Z'
anychar ::= alpha | numeric | specialchar
Comments ::= '#' { anychar }
Decimal ::= Integer '.' [ Integer ]
Float ::= Decimal ( 'e' | 'E' ) Integer
Integer ::= [ '-' ] numeric { numeric }
NodeName ::= alpha { alpha | numeric | '_' } [ ( '[' numeric ':' numeric ']' ) | ( '*' numeric '*' ) ]
numeric ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
Parameters ::= Float | Decimal | Integer
specialchar ::= '`' | '~' | '!' | '@' | '#' | '$' | '%' | '^' | '&' | '*' | '(' | ')' | '-' | '_' | '=' | '+' | '[' | ']' |
          '{' | '}' | ';' | ':' | ''' | '"' | ',' | '.' | '/' | '<' | '>' | '?'
VoltDefs ::= NodeName { Parameters }
```

Figure 19: Voltage Definition File BNF

8. ERRORS AND WARNINGS

8.1. Errors

0101 Vector Syntax At :
Should be name[#:#] as in myvector[5:0] or yourvector[3:8]

0102 Vector <name> Length Exceeds 32
The total number of nodes per vector cannot exceed 32.

0201 Node <name> Already Defined
Node Names must be unique.

0301 <vector name> NULL Node List in Vector
Only power and ground vectors can be defined without a node list.

0302 Vector exceeds 32 bits
The total number of nodes per vector cannot exceed 32.

0401 <vector name>: NULL Vector
Only power and ground vectors can be defined without a node list.

0402 Vector <vector name> Already Exists
Vectors must have unique names.

0403 Vector <vector name> Has NULL Node List
Only power and ground vectors can be defined without a node list.

0501 Bi-directional <vector name> Already Defined
Only power and ground vectors can be defined without a node list

0502 Bi-directional on <vector name> by <control node name>
Either the control node or the vector are not defined. One cause is when the bi-directional appears before the control node in the vector file (.vec)

0601 Excessive String Length
The internal string length is excessive. It may help to shorten the vector and node names.

0602 Vector Name Character.
Allowed characters are {alpha}{alphanumeric |_}*

0603 Vector Missing Type
Syntax is <vector name> <type> where type is "**POWER**", "**GROUND**", "**CLOCK**", "**INPUT**", "**OUTPUT**", "**INODE**", or "**BIDIRECTIONAL**".

0604 Unexpected EOF
Premature End of File while processing a line.

0605 Vector Type
Syntax is <vector name> <type> where type is "**POWER**", "**GROUND**", "**CLOCK**", "**INPUT**", "**OUTPUT**", "**INODE**", or "**BIDIRECTIONAL**".

0606 Power Node List
Bad Character or ill-formed node in the power node list.

0607 Unexpected EOF
Premature End of File while processing a line.

0608 Missing Radix/Nodes
Radix and Nodes are omitted from input vector.

0609 Radix Syntax
Radix must be "**BIN**", "**DEC**", or "**HEX**".

0610 Unexpected EOF
Premature End of File while processing a line.

0611 Missing Radix
Radix must be "**BIN**", "**DEC**", or "**HEX**".

O612 Radix Syntax
Radix must be "**BIN**", "**DEC**", or "**HEX**".

0613 Unexpected EOF
Premature End of File while processing a line.

0614 Missing Radix
Radix must be "**BIN**", "**DEC**", or "**HEX**".

O615 Radix Syntax
Radix must be "**BIN**", "**DEC**", or "**HEX**".

0616 Unexpected EOF
Premature End of File while processing a line.

0617 Missing Node List
Only power and ground vectors can be defined without a node list

0618 Node Syntax
Bad Character or ill-formed node in the node list.

0619 Node Syntax
Bad Character or ill-formed node in the node list.

0620 Unexpected EOF
Premature End of File while processing a line.

0621 Missing Node List
One or more nodes required.

0622 Node Syntax
Bad Character or ill-formed node in the node list.

0623 Unexpected EOF
Premature End of File while processing a line.

0624 Control Node Syntax
Bad Character or ill-formed node in the node list.

0625 Control Node Syntax
Bad Character or ill-formed node in the node list.

0626 Control Value Out of Range
Only the logic values "**0**" or "**1**" are allowed.

0627 Unexpected EOF
Premature End of File while processing a line.

0628 Control Value Syntax
Only the logic values "**0**" or "**1**" are allowed.

0629 Control Value Syntax
Only the logic values "**0**" or "**1**" are allowed.

0701 Excessive String Length
An internal string is too long. It may help to shorten the vector and node names.

0702 Node Name Character
Bad Character or ill-formed node in the node list.

0703 Pin Name Missing
<node name> <pin number> <tester channel> syntax.

0704 Unexpected EOF
Premature End of File while processing a line.

0705 Pin Name Syntax
Bad character in the pin name or ill-formed pin name.

0706 Undefined Node
Node must be defined in the vector file (.vec)

0707 Test Channel Missing
<node name> <pin number> <tester channel> syntax.

0708 Unexpected EOF
Premature End of File while processing a line.

0709 Test Channel Syntax
Channel number must be a hexadecimal digit or "x"..

0801 Excessive String Length
An internal string is too long. It may help to shorten the vector and node names.

0802 Node Name Character
Bad Character or ill-formed node in the node list.

0803 Too Many Time Entries
For a clock vector or node, 5 entries are expected (period, clock delay, clock width, rise, fall). For an input vector or node, 4 entries are expected (setup, hold, rise, fall). For an output or node vector or node, 2 entries are expected (delay and width). For a bi-directional vector or node, 6 entries are expected (setup, hold, rise, fall, delay, width).

0804 Too Many Time Entries

0805 Too Many Time Entries

0806 Too Many Time Entries

0807 Time Syntax
Any number, integer style, decimal style or exponential style (c) are acceptable.

0808 Undefined node or vector
Node or vector referenced was not defined in the vector file (.vec).

0809 Time Meaningless on Power and Ground

0810 Syntax Error: Number of Entries
Clock vector or node does not have 5 timing entries.

0811 Syntax Error: Number of Entries
Input vector or node does not have 4 timing entries.

0812 Syntax Error: Number of Entries
Output or node vector or node does not have 2 timing entries.

0813 Syntax Error: Number of Entries
Bi-directional vector or node does not have 6 timing entries.

1101 Multiple Periods Not Allowed
All clocks must have the same period.

1102 Multiple Clock Delays Not Allowed
All clocks must have the same first edge delay.

1103 Multiple Clock Widths Not Allowed
All clocks must have the same pulse width.

1301 Ground Not Found
No Ground Node Specified.

1302 Power Not Found
No Power Node Specified.

1303 Missing Vector from Link List
The initialization call (Init_SOAR) vector link list is missing one or more vectors.

1304 Bi-directional <vector name> Control Error
Internal Error.

1401 Bi-directional Control Node <node name> is Illegal Type
Only **INPUT**, **OUTPUT**, and **INODE** type nodes can be used to control bi-directional vectors.

1402 Vector Not Found
Link indicates a vector not defined in the .vec file.

8.2. Warning Summary

0601 Initial Value No Longer Supported-Ignored

0801 Vector Selected over Node Name
When both a vector and a node have the same name, timing values are assigned to the entire vector.

1101 No Clocks Defined
Default Timing will be used for "cycles." To coerce this timing, use the special vector and node named **DUMMY_CLOCK**. In other words, put the following in the .vec file:
DUMMY_CLOCK CLOCK
DUMMY_CLOCK

1301 Input Vector <vector name> Not Initialized->Coerced
An undefined input was specified. This is defined now by default to whatever value (correct or not) set by the calling routine.

1401 Using Default Power Setting
Rather than a user specified voltage setting, the default is being used.

1402 Multiple Clocks, First Period Being Used
Bad practice, but HSPICE allows clocks with different settings. However, all setup and delay times are calculated from the first clock specified in the .vec file.

1403 Input Node <node name> Not Initialized->Coerced
Input node was not defined by calling program. It is set as defined and whatever value was set is used.

9. Sample Programs

Three sample circuits are described in this section. The operation of each circuit and the test sequence for each circuit is described. A sample C code routine and simulation output is shown. Extracts from the appropriate conversion output files are also shown.

The three circuits illustrate the methods described in this document. The Meuller C-Element is an asynchronous building block. The circuit contains storage, but without a clock. RAM is a common asynchronous circuit. This illustrates the absence of a clock. The FIFO is a synchronous example circuit. The RAM and the FIFO, although small in the examples, illustrate the complex simulation and test files that can be created.

9.1. Meuller C-Element

The Meuller C-Element (see Figure 20) is a storage element used in join operations for asynchronous circuits. The essence of the circuit is best shown in the truth table, Table 6. The state changes when the two inputs agree. A simulation of this truth table is generated with the C-code of Figure 21. A single word was changed to generate the HSPICE conversion, VIEWLOGIC to HSPICE.

Table 6: C Element Truth Table

IN0	IN1	$OUT0_{T=1}$
0	0	0
0	1	$OUT0_{T=0}$
1	0	$OUT0_{T=0}$
1	1	1

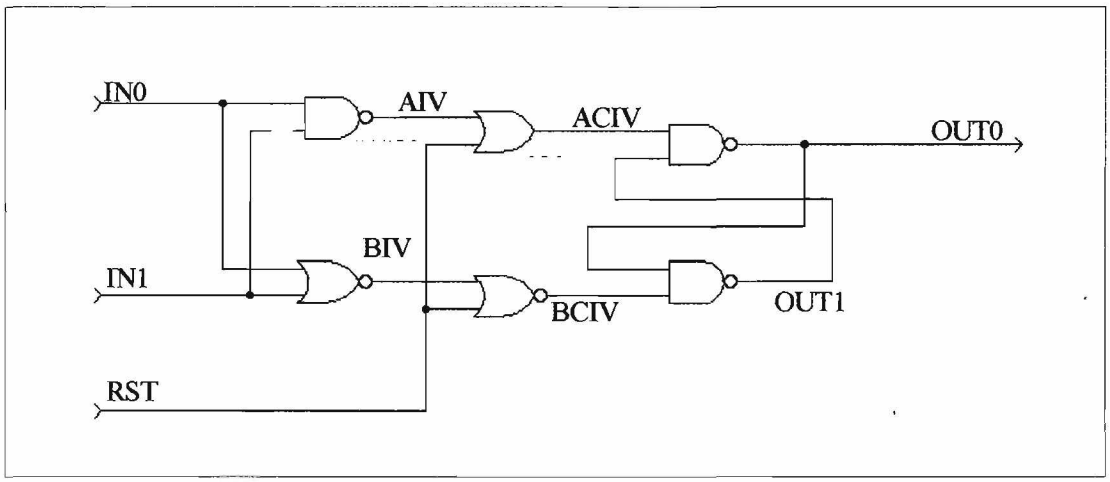


Figure 20: C Element Logic Schematic

```
/*****************************************************************
/*****************************************************************
**          Pin./Vector          Type          Description
**          VDD                  POWER         Power Pin
**          GND                  GROUND        Ground Pin
**          DUMMY_CLOCK          CLOCK         To set my own timing parameters
**          IN                   INPUT         Two Inputs
**          RST                  INPUT         Clear Signal
**          OUT                  OUTPUT        Two Outputs
**          AI                   INODE         Internal
**          BI                   INODE         Internal
**          ACI                  INODE         Internal
**          BCI                  INODE         Internal
*/
#include "convers.h"
unsigned long CLK = 0x1;                       /* High-Going Dummy Clock */
unsigned long RST = 0x1;                       /* High Active Reset */
unsigned long IN = 0x3;                        /* Inputs */
unsigned long OUT = 0x1;                       /* Outputs */
unsigned long AI = 0x0;                        /* Internal */
unsigned long BI = 0x0;                        /* Internal */
unsigned long ACI = 0x0;                       /* Internal */
unsigned long BCI = 0x0;                       /* Internal */
```

Figure 21: C Element C Code

```
VECTOR_LINK bci = { "BCIV", &BCI, TRUE, NULL };
VECTOR_LINK aci = { "ACIV", &ACI, TRUE, &bci };
VECTOR_LINK bi = { "BIV", &BI, TRUE, &aci };
VECTOR_LINK ai = { "AIV", &AI, TRUE, &bi };
VECTOR_LINK out = { "OUT", &OUT, TRUE, &ai };
VECTOR_LINK in = { "IN", &IN, TRUE, &out };
VECTOR_LINK rst = { "RST", &RST, TRUE, &in };
VECTOR_LINK clk = { "DUMMY_CLOCK", &CLK, TRUE, &rst };
/* */
void Resolve_values ( void );
{
        int temp;
        AI = ( IN != 3 );
        BI = ( IN == 0 );
        ACI = ( AI | RST );
        BCI = ( ( BI | RST ) == 0 );
        temp = ( OUT << 2 ) | ( ACI << 1 ) | BCI;
        switch ( temp )
            {
                case 0x0:                                   /* Y = 0, YB = 0, ACI = 0, BCI = 0, Bad */
                case 0x1:                                   /* Y = 0, YB = 0, ACI = 0, BCI = 1, Bad */
                case 0x2:                                   /* Y = 0, YB = 0, ACI = 1, BCI = 0, Bad */
                case 0x3:                                   /* Y = 0, YB = 0, ACI = 1, BCI = 1, Bad */
                case 0x4:                                   /* Y = 0, YB = 1, ACI = 0, BCI = 0, Bad */
                case 0x8:                                   /* Y = 1, YB = 0, ACI = 0, BCI = 0, Bad */
                case 0xc:                                   /* Y = 1, YB = 1, ACI = 0, BCI = 0, Bad */
                case 0xf:                                   /* Y = 1, YB = 1, ACI = 1, BCI = 1, BAD */
                    printf ( "Illegal State: %x reached\n", temp );
                    break;
                case 0x5:                                   /* Y = 0, YB = 1, ACI = 0, BCI = 1, to Y */
                case 0xd:                                   /* Y = 1, YB = 1, ACI = 0, BCI = 1, to Y */
                    OUT = 2;
                    break;
                case 0xa:                                   /* Y = 1, YB = 0, ACI = 1, BCI = 0, to YB */
                case 0xe:                                   /* Y = 1, YB = 1, ACI = 1, BCI = 0, to YB */
                    OUT = 1;
                    break;
                case 0x6:                                   /* Y = 0, YB = 1, ACI = 1, BCI = 0, Stay */
                case 0x7:                                   /* Y = 0, YB = 1, ACI = 1, BCI = 1, Stay */
                case 0x9:                                   /* Y = 1, YB = 0, ACI = 0, BCI = 1, Stay */
                case 0xb:                                   /* Y = 1, YB = 0, ACI = 1, BCI = 1, Stay */
                    break;
            }
}
```

Figure 21 continued: C Element C Code-Circuit Resolution

```
int main ()
{
    Init_SOAR ( "celement", &clk, SCMOS, VIEWLOGIC );   /* Start the Conversion (in reset) */
    IN = 0;                                             /* Remove Input 1's */
    Resolve_Values ();
    SOAR ( "World Is Reset Cycle" );
    RST = 0;                                            /* Finish Reset */
    Resolve_Values ();
    SOAR ( "Finish Reset" );
    IN = 1;                                             /* switch to 01 to 10, look for hazard */
    Resolve_Values ();
    SOAR ( "Preset for Hazard" );
    IN = 2;
    Resolve_Values ();
    SOAR ( "Look for Hazard" );
    IN = 0;
    Resolve_Values ();
    SOAR ( "Back to Zero" );
    IN = 2;                                             /* switch to 10 to 0,1 look for hazard */
    Resolve_Values ();
    SOAR ( "Preset for Hazard" );
    IN = 1;
    Resolve_Values ();
    SOAR ( "Look for Hazard" );
    IN = 0;
    Resolve_Values ();
    SOAR ( "Back to Zero" );
    IN = 1;                                             /* Normal Operation */
    Resolve_Values ();
    SOAR ( "Preset Normal One" );
    IN = 3;
    Resolve_Values ();
    SOAR ( "Do Normal One" );
    IN = 2;
    Resolve_Values ();
    SOAR ( "Preset Normal Zero" );
    IN = 0;
    Resolve_Values ();
    SOAR ( "Do Normal Zero" );
    Finish_SOAR( );
    return 0;
}
```

Figure 21 continued: C Element C Code-Main Function

The C element circuit was simulated with both ViewSim and HSPICE. One SOAR event, or cycle, command file generated by the conversion is shown in Figure 22. The HSPICE stimulus generated is shown in Figure 23. The simulation results are shown in Figures 24 and 25 respectively.

```
|     Circuit Definitions at the beginning
logfile celement.log
ticksize   1.00ps
net -vsm celement.vsm
|     Vector Defintions
vector RST RST
( radix bin RST; )
vector IN IN1 IN0
( radix bin IN; )
vector OUT OUT1 OUT0
( radix bin OUT; )
vector AIV AI
( radix bin AIV; )
vector BIV BI
( radix bin BIV; )
vector ACIV ACI
( radix bin ACIV; )
vector BCIV BCI
( radix bin BCIV; )
|     Set Signals to Watch
wave celement.wfm CLR IN OUT AIV BIV ACIV BCIV
watch CLR IN OUT AIV BIV ACIV BCIV
trace OUT AIV BIV ACIV BCIV
|     A Sample Period Follows
|     Period 1
|     World Is Reset Cycle
assign IN 00\B
sim   3.00ns
check AIV 1\B
check BIV 1\B
sim   3.00ns
check ACIV 1\B
check BCIV 0\B
sim   3.00ns
check OUT 01\B
sim   1.00ns
check ACIV 1\B
check BCIV 0\B
check OUT 01\B
check AIV 1\B
check BIV 1\B
|     Finish the Simulation
logfile
quit
```

Figure 22: Partial C Element Command File

```
*       Options and Globals Are Setup
.OPTIONS NOMOD POST PROBE
V0 VDD GND 5.00V
*       Clock Parameters
.PARAM Tper0=  10.00ns
.PARAM Tcd0=   2.00ns
.PARAM Tcpw0=   5.00ns
.PARAM Tcr0= 300.00ps
.PARAM Tcf0= 300.00ps
*       Part of the Comment Table Generated
*       NO BIDIRECTIONAL PINS
*       PERIOD COMMENTS
*       Period: 1: World Is Reset Cycle
*       Period: 2: Finish Reset
*
*       Piece-Wise Linear Supplies
VRST RST GND PWL ( 0ns    4.00v
+         '2 * Tper0 +    0.00'   4.00v
+         '2 * Tper0 +  100.00p'  400.00mv )
VIN0 IN0 GND PWL ( 0ns    4.00v
+         '1 * Tper0 +    0.00'   4.00v
+         '1 * Tper0 +  100.00p'  400.00mv
+         '4 * Tper0 +    0.00'  400.00mv
+         '4 * Tper0 +  100.00p'    4.00v
+         '5 * Tper0 +    0.00'   4.00v
+         '5 * Tper0 +  100.00p'  400.00mv
+         '6 * Tper0 +    0.00'  400.00mv
+         '6 * Tper0 +  100.00p'    4.00v
+         '7 * Tper0 +    0.00'   4.00v
+         '7 * Tper0 +  100.00p'  400.00mv
+         '10 * Tper0 +   0.00'  400.00mv
+         '10 * Tper0 +  100.00p'   4.00v
+         '12 * Tper0 +   0.00'   4.00v
+         '12 * Tper0 +  100.00p'  400.00mv )
*       Probe, Transient and End
.PROBE I(VVDD) V(RST) V(IN0) V(IN1) V(OUT0) V(OUT1)
.TRAN  100.00fs '13 * Tper0'
.END
```

Figure 23: Partial C Element HPSICE Stimulus File

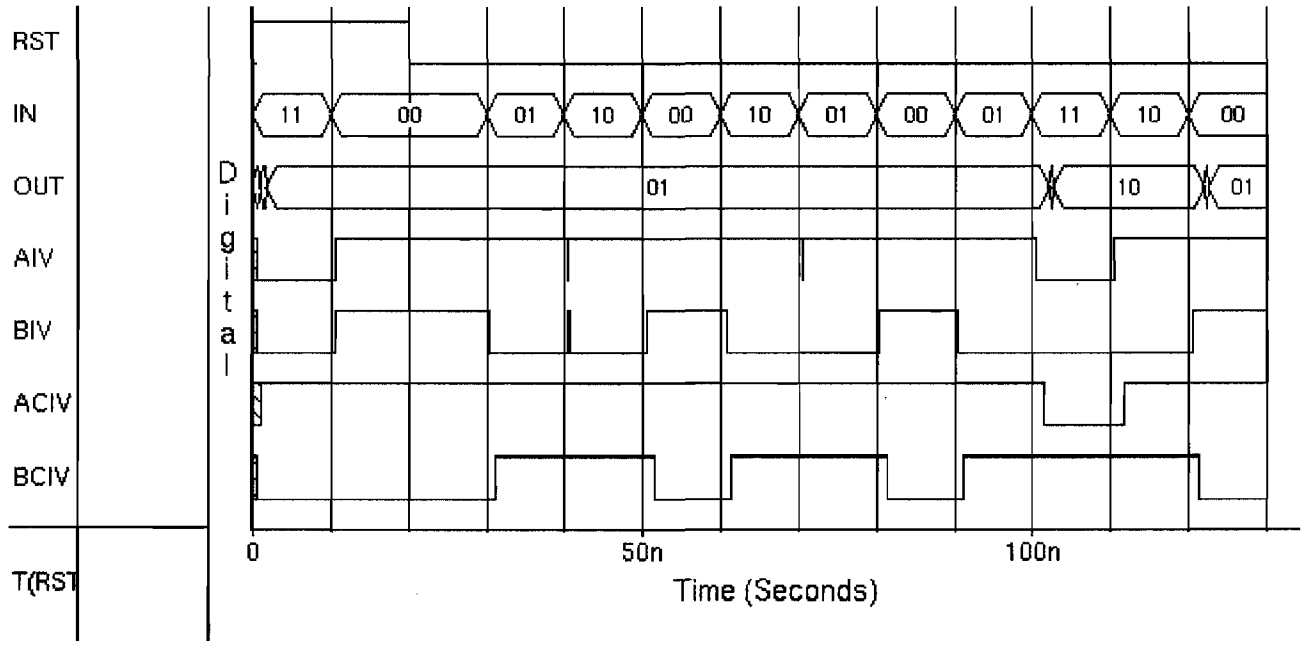


Figure 24: C Element ViewSim Results

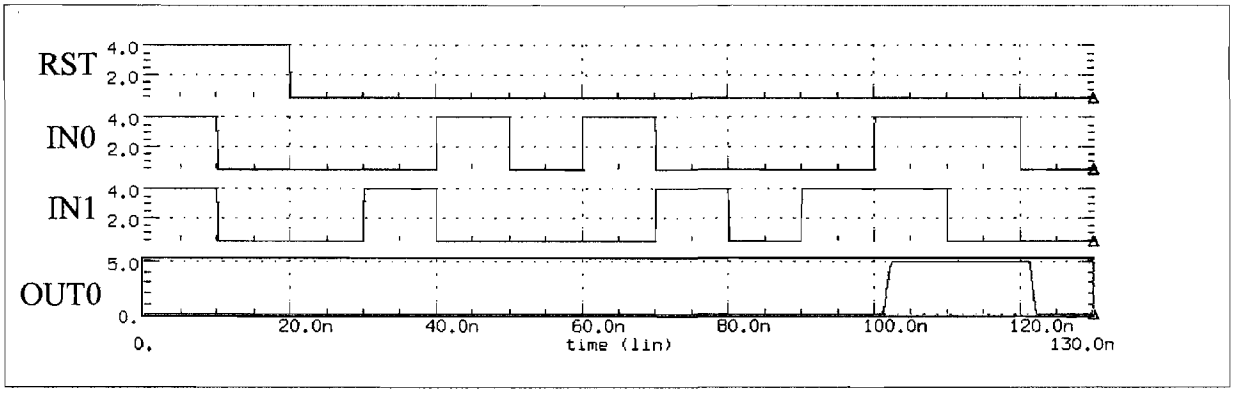


Figure 25: C Element HSPICE Results

Conversion of the C Element to generate the simulations shown required two definition files. The vector definition file shown in Figure 26 is always required. To alter the default timing, a timing definition file was used. This file can be found in Figure 27.

```
# Vector Definition File for the C Element
VDD             POWER
GND             GROUND
DUMMY_CLOCK     CLOCK           DUMMY_CLOCK
RST             INPUT   BIN     RST
IN              INPUT   BIN     IN[1:0]
OUT             OUTPUT  BIN     OUT[1:0]
AIV             INODE   BIN     AIV
BIV             INODE   BIN     BIV
ACIV            INODE   BIN     ACIV
BCIV            INODE   BIN     BCIV
```

Figure 26: C Element Vector Definition File

```
# Timing Definition File fo the C Element
#               Period  Delay   Width   Rise    Fall
DUMMY_CLOCK     10e-9   2e-9    5e-9    3e-10   3e-10
#               Setup   Hold    Rise    Fall
RST             2e-9    -2e-9   1e-10   1e-10
IN              2e-9    -2e-9   1e-10   1e-10
#               Delay   Width
OUT             7e-9    1e-9
AIV             1e-9    7e-9
BIV             1e-9    7e-9
ACIV            4e-9    4e-9
BCIV            4e-9    4e-9
```

Figure 27: C Element Timing Definition File

9.2. RAM

The RAM circuit is organized as eight bits by eight words. It is a typical 6T, high-speed RAM. The control lines are read/write and output enable. The pseudo-code for the simulation pattern is shown in Figure 28. The pattern writes a "walking one" into the RAM and then reads it back for comparison.

The vector definition file was, as always, supplied. A timing definition file was also supplied to generate the NRZ timing, avoiding false writes. The C Code RAM emulation was converted to both HSPICE stimulus and ViewSim command output. Figure 29 shows the ViewSim results.

```
Define Variables
Define Vector Links
OE Bar <= 1; R/W <= 1; Data IN <= 0x01;
Init_SOAR Call
For Each Address Do
        Pulse R/W Low in SOAR Calls
        Shift Data IN Left
OE Bar <= 0; Data OUT <= 0x01;
For Each Address Do
        SOAR Event To Read Data
Finish_SOAR Call
```

Figure 28: RAM Simulation Pseudo-Code

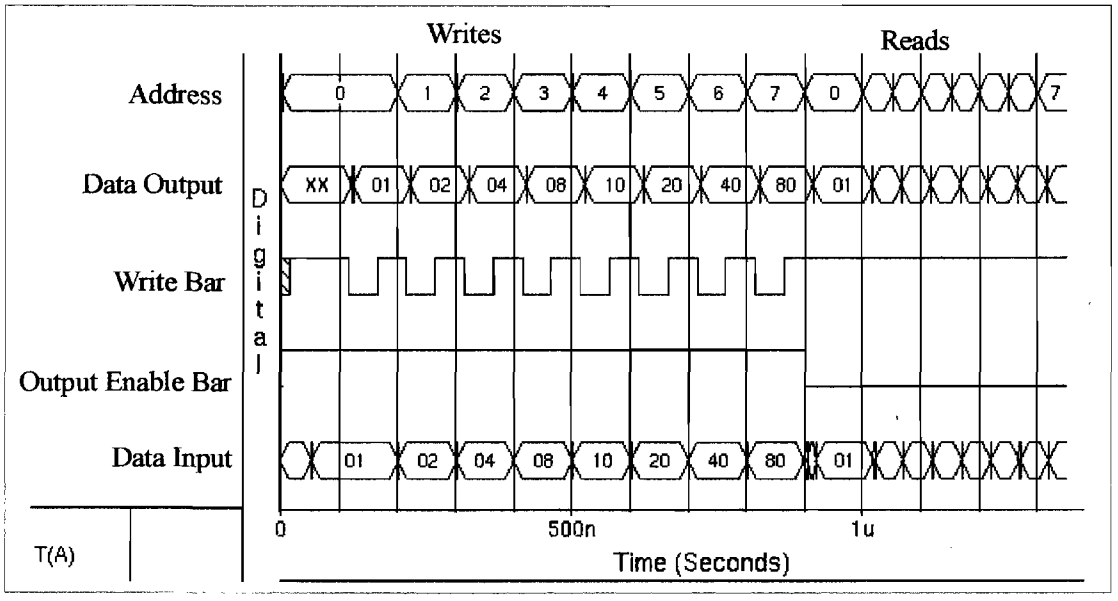


Figure 29: RAM ViewSim Results

Conversion of the RAM is straightforward. The definition files use the NRZ format similar to the C Element The advantage of the method is clearly seem in the pseudo-code. This code is complete, independent of the RAM size.

9.3. FIFO

The FIFO is a four word by eight bit FIFO using a clocked shifting architecture. The architecture, shown in Figure 30, is used to provide high throughput and low latency in a single clock environment. This circuit was simulated with ViewSim and HSPICE, manufactured and tested on the LV500.

The simulation and test pattern used is listed in pseudo-code form in Figure 31. The pattern loads the entire FIFO, shifts all four words out, loads two words, performs multiple shift/load operations, and finally checks the SCAN path. The HSPICE results of the load, shift and load/shift tests are shown in Figure 32.

Vector definition, pin definition and timing definition files were used to simulate and test this circuit. They are shown in Figures 33, 34, and 35

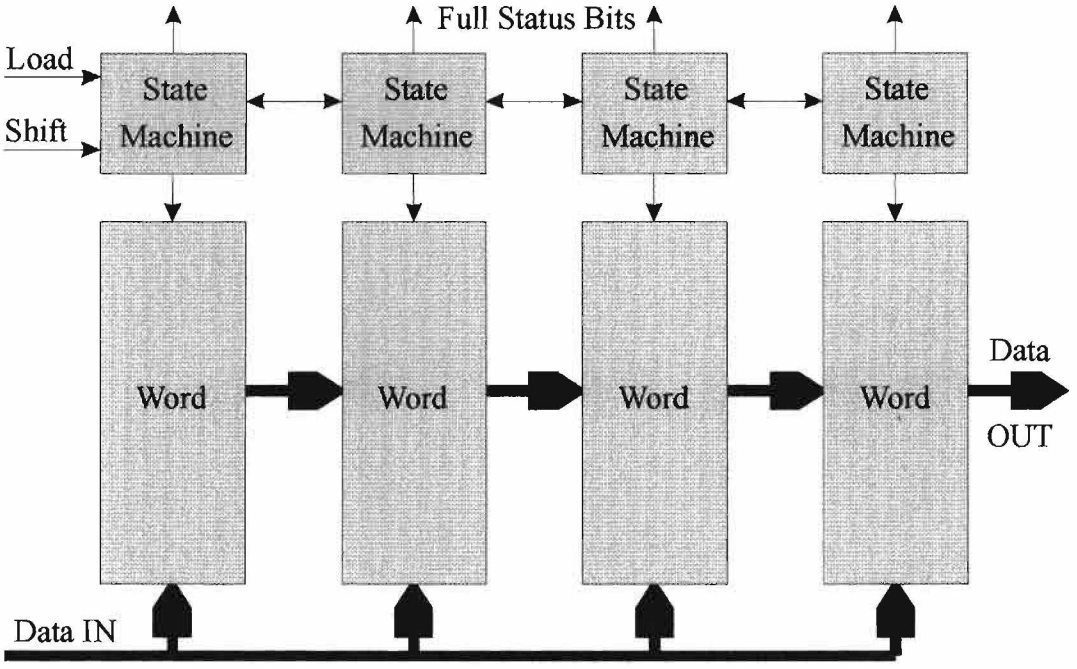


Figure 30: FIFO Architecture

```
Define Data Values and Links
SHIFT <= 0; LOAD <= 1; CLR <=1; DATA IN<=0x55;
Init_SOAR Call
For Word Count Do
        Load Data (SOAR CALL)
        Complement Data IN
LOAD <= 0; Data OUT = 0x55;
For Word Count Do
        Shift Data (SOAR Call)
        Complement Data OUT
SHIFT <= 0; LOAD <= 1; Data IN <= 0;
Load One Location (SOAR Call)
Data IN <= 1;
Load a Second Location (SOAR Call )
Shift = 1; Data IN <= 2;
15 Times Do
        Load and Shift (SOAR Call)
        Data IN <= Data IN + 1
Test <= 1;
Load Alternating 0/1 into scan path (36 SOAR Calls)
Read Alternating 0/1 from scan path (36 SOAR Calls)
```

Figure 31: FIFO Simulation and Test Generation Pseudo-Code

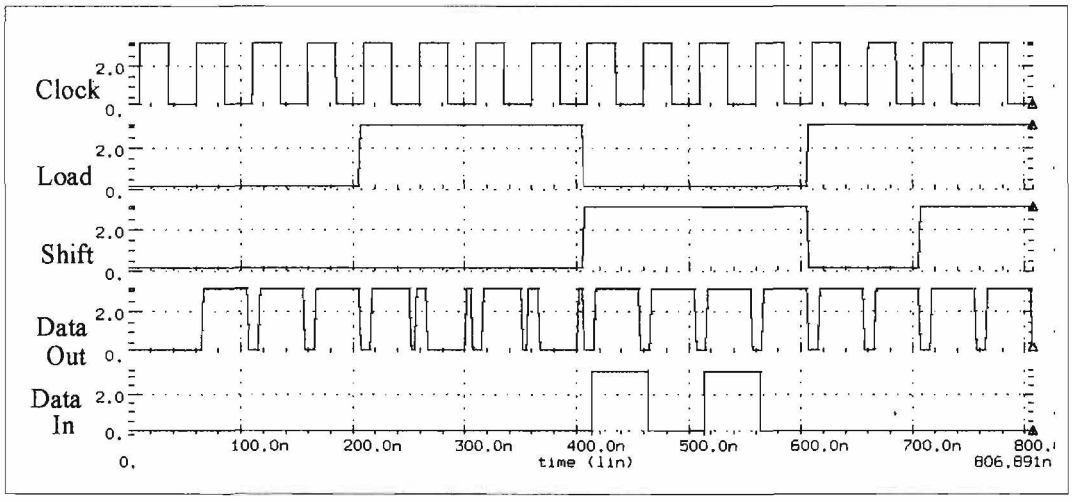


Figure 32: FIFO Simulation Results

# Vector Definition File for the FIFO					
VDD	POWER				AVDDA AVDDB VDDA VDDB AVDDS VDDS
GND	GROUND				AVSSA AVSSB GNDA GNDB AVSSS
CLK	CLOCK				PAD_CLK
DIN	DIN	HEX			PAD_DIN[0:7]
CONTROL	INPUT	BIN			PAD_TEST PAD_ENABLE PAD_SHIFT PAD_LOAD PAD_RST_L
DOUT	OUTPUT	HEX			PAD_DOUT[0:7]
STATUS	OUTPUT	BIN			PAD_ERROR
FULL1	BIDIRECTIONAL	BIN	PAD_ENABLE	1	PAD_FULL[0:1]
FULL2	BIDIRECTIONAL	BIN	PAD_ENABLE	1	PAD_FULL[2:3]
SIN	INPUT	BIN			PAD_SCANIN
SOUT	OUTPUT	BIN			PAD_SCANOUT

Figure 33: FIFO Vector Definition File

# Node Definition File					
AVDDS			PAD_DIN*0*	15	1.c
AVDDA	20	x.x	PAD_TEST	18	2.2
AVDDB	39	x.x	PAD_SCANIN	17	2.0
VDDS			PAD_ENABLE	19	2.4
VDDA	10	x.x	PAD_RST_L	24	2.1
VDDB	30	x.x	PAD_SHIFT	22	2.5
AVSSS			PAD_LOAD	23	2.3
AVSSA	21	x.x	PAD_DOUT*7*	35	0.b
AVSSB	40	x.x	PAD_DOUT*6*	34	0.d
GNDS			PAD_DOUT*5*	33	0.f
GNDA	11	x.x	PAD_DOUT*4*	32	1.1
GNDB	21	x.x	PAD_DOUT*3*	29	1.7
PAD_CLK	16	1.e	PAD_DOUT*2*	28	1.9
PAD_DIN*7*	6	0.a	PAD_DOUT*1*	27	1.b
PAD_DIN*6*	7	0.c	PAD_DOUT*0*	26	1.d
PAD_DIN*5*	8	0.e	PAD_SCANOUT	4	0.6
PAD_DIN*4*	9	1.0	PAD_ERROR	3	0.4
PAD_DIN*3*	12	1.6	PAD_FULL*0*	37	0.7
PAD_DIN*2*	13	1.8	PAD_FULL*1*	38	0.5
PAD_DIN*1*	14	1.a	PAD_FULL*2*	1	0.0
			PAD_FULL*3*	2	0.2

Figure 34: FIFO Pin Definition File

# Timing Definition File for the FIFO						
# Clock Signals	Period	Delay	Width	Rise	Fall	
CLK	50e-9	10e-9	25e-9	2e-10	2e-10	
# Input Signals	Setup	Hold	Rise	Fall		
DIN	5e-9	5e-9	2e-9	2e-9		
CONTROL	5e-9	-5e-9	2e-9	2e-9		
SIN	5e-9	-5e-9	2e-9	2e-9		
# Output Signals	Delay	Width				
DOUT	25e-9	8e-9				
STATUS	25e-9	8e-9				
SOUT	25e-9	8e-9				
# Bi-directional Signals	Setup	Hold	Rise	Fall	Delay	Width
FULL1	5e-9	-5e-9	2e-9	2e-9	25e-9	8e-9
FULL2	5e-9	-5e-9	2e-9	2e-9	25e-9	8e-9

Figure 35: FIFO Timing Definition File

10. BIBLIOGRAPHY

[1] EDIF TEST TECHNICAL SUBCOMMITTEE, *EDIF Electronic Design Interchange Format 2 0 300*, Edited by M. Wahl, 1992.

[2] META SOFTWARE, *HSPICE User's Manual: Volume I, Simulation and Analysis, Version 96.1*, META-Software, Incorporated, Campbell, California, February 1996.

[3] META SOFTWARE, *HSPICE User's Manual: Volume II, Elements and Device Models, Version 96.1*, META-Software, Incorporated, Campbell, California, February 1996.

[4] META SOFTWARE, *HSPICE User's Manual: Volume III, Applications and Examples, Version 96.1*, META-Software, Incorporated, Campbell, California, February 1996.

[5] TEKTRONIX INCORPORATED, *LV500 Operator's Manual*, Tektronix, Incorporated, Beaverton, Oregon, September 1989.

[6] VIEWLOGIC CORPORATION, *ViewSim Reference Manual*, Viewlogic Corporation, 1989.